Actividad: Mejorando la Página de Películas

En esta actividad, mejoraremos nuestra aplicación de películas implementando nuevas funcionalidades:

- Crearemos una página de detalles para cada película.
- Actualizaremos la página de películas populares para hacerla más dinámica.
- Construiremos componentes reutilizables y nuevos servicios para acceder a diferentes endpoints de la API.
- Seguiremos un enfoque paso a paso para guiar el desarrollo.
- Finalizaremos con una actividad libre en la que deberán aplicar lo aprendido sin instrucciones detalladas.

Utilizaremos React y Next.js con el objetivo de practicar buenas prácticas como modularización, reutilización de componentes y trabajo con datos externos.

by Cesar Betancourt

1. Crear el Servicio getMovieById

Comenzaremos creando un nuevo servicio para obtener los detalles de una película específica por su ID. Este servicio será crucial para nuestra página de detalles de película.

Crea un nuevo archivo llamado **getMovieById.ts** en la carpeta **src/services/movies** con el siguiente contenido:

```
import api from "../api";

export const getMovieById = async (id: string) => {
  try {
    const { data } = await api.get(`/movie/${id}`);
    return data;
  } catch (error) {
    throw error;
  }
};
```

Este servicio utilizará la API que ya hemos configurado para hacer una solicitud GET a la ruta **/movie/{id}**, donde {id} es el ID de la película que queremos obtener.

2. Crear la Página de Detalles de Película

Ahora, crearemos una nueva página que mostrará los detalles de una película específica. Esta página utilizará el servicio que acabamos de crear.

Crea un nuevo archivo llamado **page.tsx** en la carpeta **src/app/movie/[id]** con el siguiente contenido:

```
// src/app/movie/[id]/page.tsx
"use client";

import { useEffect, useState } from "react";
import { useParams, useSearchParams } from "next/navigation";
import { getMovieById } from "@/services/movies/getMovieById";

const MovieDetailPage = () => {
  const { id } = useParams(); // id is a string | string[] | undefined
  const searchParams = useSearchParams();
  const from = searchParams.get("from");
  console.log(from);
  const [movie, setMovie] = useState<any>(); // pendiente de type
  const [loading, setLoading] = useState<boolean>(true);
  const [error, setError] = useState<string | null>(null);

  useEffect(() => {
    if (!id || typeof id !== "string") return;

    const fetchMovie = async () => {
      setLoading(true);
      try {
        const data = await getMovieById(id);
        setMovie(data);
      } catch (err) {
        console.error("Error fetching movie", err);
        setError("Could not load movie.");
      } finally {
        setLoading(false);
      }
    };

    fetchMovie();
  }, [id]);

  if (loading) return <div>Loading movie...</div>;
  if (error) return <div>{error}</div>;
  if (!movie) return <div>No movie found.</div>;

  return (
    <div>
      <h1 className="text-2xl font-bold mb-2">{movie.title}</h1>
      <p className="text-gray-700">{movie.overview}</p>
      {/* Add more movie details here */}
    </div>
  );
};

export default MovieDetailPage;
```

Este componente `MovieDetailPage` en Next.js:

1. Lee el `id` de la URL usando `useParams()` — por ejemplo, si estás en `/movie/123`, el `id` será `"123"`.
2. Lee parámetros de búsqueda (query params) como `?from=popular` usando `useSearchParams()`.
3. Hace una petición a la función `getMovieById(id)` (un servicio que imagino consulta la API de películas) para traer la información del detalle de esa película.
4. Muestra distintos estados:
   - "Loading movie..." mientras carga.
   - Un error si ocurre.
   - Un mensaje de "No movie found" si no encuentra la película.
   - Finalmente muestra el título y la descripción de la película.

Todo esto se maneja usando `useState` y `useEffect` para controlar la carga (`loading`), el error (`error`) y el dato (`movie`).

¿Por qué en Next.js las páginas se llaman `[id]`?

En Next.js los corchetes (`[id]`) en una carpeta o archivo significan una ruta dinámica.

- Por ejemplo, si tienes `src/app/movie/[id]/page.tsx`, cuando alguien visite `/movie/123` o `/movie/456`, Next.js entiende que `id` es una variable.
- Es como decirle al framework: "cualquier cosa que venga aquí, guárdala como una variable que puedo usar en mi componente".
- Eso es lo que luego capturas con `useParams()`.

3. Actualizar la Página de Películas Populares

Ahora que tenemos una página de detalles de película, necesitamos actualizar nuestra página de películas populares para que los usuarios puedan navegar a la página de detalles al hacer clic en una película.

Actualiza el archivo **src/app/popular/page.tsx** para incluir enlaces a la página de detalles de cada película:

```
import Link from "next/link";

// ... (código existente) ...
// Mostrar las peliculas
  return (
<div>
  <h3 className="text-3xl font-bold mb-6">Popular Movies</h3>

  {/* Loading indicator */}
  {loading && <h5 className="text-lg text-gray-500">Cargando...</h5>}

  {/* Grid Layout */}
  <div className="grid grid-cols-1 sm:grid-cols-2 md:grid-cols-4 lg:grid-cols-5 gap-6">
   {movies?.map((movie) => (
     <Link
       key={movie.id}
       href={{
         pathname: `/movie/${movie.id}`,
         query: { from: "popular" },
       }}
       >
         {movie.title}
       </Link>
     ))}
    </div>
  </div>
 );

// ... (código existente) ...
```

Este cambio envolverá cada película en un componente Link que dirigirá al usuario a la página de detalles de la película cuando se haga clic en ella.

4. Configurar Next.js para Imágenes Externas

Para mostrar imágenes de películas desde una fuente externa, necesitamos configurar Next.js para que permita estas imágenes. Actualizaremos la configuración de Next.js y crearemos un archivo de configuración para nuestra aplicación.

Actualiza el archivo **next.config.js** en la raíz del proyecto:

```
import type { NextConfig } from "next";

const nextConfig: NextConfig = {
  images: {
    domains: ["image.tmdb.org"],
  },
};

export default nextConfig;
```

Luego, en el archivo index.ts en la carpeta src/Config con el siguiente contenido:

```
const Config = {
  API_URL: "https://api.themoviedb.org/3",
  IMAGE_SOURCE: "https://image.tmdb.org/t/p/w500"
};

export default Config;
```

Estos cambios permitirán que Next.js cargue imágenes desde el dominio de TMDB y proporcionarán una configuración para las imágenes.

5. Crear el Componente MovieCard

Para mejorar la presentación de nuestras películas, crearemos un componente MovieCard que mostrará la información de cada película de manera más atractiva.

Crea un nuevo archivo llamado **MovieCard.tsx** en la carpeta **src/components/MovieCard**, puede hacer el componente a tu gusto o con el siguiente contenido:

```
import Config from "@/config";
import Image from "next/image";

interface IMovieCard {
  title: string;
  voteAverage: number;
  posterPath: string;
  releaseYear: number;
  description: string;
}
const MovieCard: React.FC<IMovieCard> = ({
  title,
  voteAverage,
  posterPath,
  releaseYear,
  description,
}) => {
  // const router = useRouter();
  const poster = Config.IMAGE_SOURCE + posterPath;

  // You can use this to navigate instead of Link in the page.tsx
  /* const navigateMovies = (id: number) => {
    router.push(`/movie/${id}`);
  }; */

  return (
    <div className="flex items-center justify-center">
      <div className="mx-auto bg-white rounded-3xl shadow-xl">
        <div className="grid rounded-3xl max-w-[360px] shadow-sm bg-slate-100 flex-col group">
          {/* Poster Image */}
          <Image
            src={poster}
            width="360"
            height="200"
            className="rounded-t-3xl justify-center grid object-cover"
            alt={title}
          />

          <div className="p-5 z-10">
            {/* Movie Title */}

            <p className="h-10">{title}</p>
            <p className="text-slate-400 pt-2 font-semibold ">
              ({releaseYear})
            </p>

            {/* Movie Description */}
            <div className="h-20">
              <span className="line-clamp-3 py-2 h-20 text-sm font-light leading-relaxed">
                {description}
              </span>
            </div>

            {/* Movie Rating */}
            <div className="grid-cols-2 flex justify-between">
              <div className="font-black flex flex-col">
                <span className="text-yellow-500 text-xl">SCORE</span>
                <span className="text-3xl flex gap-x-1 items-center group-hover:text-yellow-600">
                  {voteAverage.toFixed(1)}
                  <svg
                    width="24px"
                    height="24px"
                    viewBox="0 0 24 24"
                    fill="none"
                    xmlns="http://www.w3.org/2000/svg"
                  >
                    <g id="SVGRepo_bgCarrier" strokeWidth="0" />
                    <g
                      id="SVGRepo_tracerCarrier"
                      strokeLinecap="round"
                      strokeLinejoin="round"
                    />
                    <g id="SVGRepo_iconCarrier">
                      <path
                        d="M9.15316 5.40838C10.4198 3.13613 11.0531 2 12 2C12.9469 2 13.5802 3.13612 14.8468 5.40837L15.1745 5.99623C15.5345 6.64193 15.7144 6.96479 15.9951 7.17781C16.2757 7.39083 16.6251 7.4699 17.3241 7.62805L17.9605 7.77203C20.4201 8.32856 21.65 8.60682 21.9426 9.54773C22.2352 10.4886 21.3968 11.4691 19.7199 13.4299L19.2861 13.9372C18.8096 14.4944 18.5713 14.773 18.4641 15.1177C18.357 15.4624 18.393 15.8341 18.465 16.5776L18.5306 17.2544C18.7841 19.8706 18.9109 21.1787 18.1449 21.7602C17.3788 22.3417 16.2273 21.8115 13.9243 20.7512L13.3285 20.4768C12.6741 20.1755 12.3469 20.0248 12 20.0248C11.6531 20.0248 11.3259 20.1755 10.6715 20.4768L10.0757 20.7512C7.77268 21.8115 6.62118 22.3417 5.85515 21.7602C5.08912 21.1787 5.21588 19.8706 5.4694 17.2544L5.53498 16.5776C5.60703 15.8341 5.64305 15.4624 5.53586 15.1177C5.42868 14.773 5.19043 14.4944 4.71392 13.9372L4.2801 13.4299C2.60325 11.4691 1.76482 10.4886 2.05742 9.54773C2.35002 8.60682 3.57986 8.32856 6.03954 7.77203L6.67589 7.62805C7.37485 7.4699 7.72433 7.39083 8.00494 7.17781C8.28555 6.96479 8.46553 6.64194 8.82547 5.99623L9.15316 5.40838Z"
                        fill="#eab308"
                      />
                    </g>
                  </svg>
                </span>
              </div>
            </div>
          </div>
        </div>
      </div>
    </div>
  );
};

export default MovieCard;
```

Este componente creará una tarjeta atractiva para cada película, mostrando su póster, título, año de lanzamiento, descripción y puntuación.

6. Actualizar la Página de Películas Populares con MovieCard

Ahora que tenemos nuestro componente MovieCard, actualizen la página de películas populares para utilizarlo.

Actualiza el archivo **src/app/popular/page.tsx**

Esta actualización reemplazará la lista simple de películas con una cuadrícula de tarjetas de películas, cada una enlazada a su página de detalles correspondiente.

7. Prueba y Depuración

Una vez que hayas implementado todos estos cambios, es hora de probar tu aplicación y asegurarte de que todo funcione correctamente.

1. Inicia tu servidor de desarrollo con **npm run dev** o **yarn dev**.
2. Navega a la página de películas populares y verifica que las tarjetas de películas se muestren correctamente con el componente de MovieCard y en una cuadricula
3. Haz clic en una película y asegúrate de que te lleve a la página de detalles de esa película.
4. Verifica que la página de detalles muestre la información correcta de la película.
5. Prueba la navegación de vuelta a la página de películas populares.

Si encuentras algún error, utiliza las herramientas de desarrollo del navegador para inspeccionar la consola y la red, y depura tu código según sea necesario.

8. Types

Tipos para el detalle de una película, utilizar IMovieDetail en src/app/movie/[id]/page.tsx en lugar de any. Crear un nuevo archivo MovieDetail.ts en la carpeta de src/types

```
export interface IMovieDetail {
  adult: boolean;
  backdrop_path: string;
  belongs_to_collection: BelongsToCollection;
  budget: number;
  genres: Genre[];
  homepage: string;
  id: number;
  imdb_id: string;
  origin_country: string[];
  original_language: string;
  original_title: string;
  overview: string;
  popularity: number;
  poster_path: string;
  production_companies: ProductionCompany[];
  production_countries: ProductionCountry[];
  release_date: Date;
  revenue: number;
  runtime: number;
  spoken_languages: SpokenLanguage[];
  status: string;
  tagline: string;
  title: string;
  video: boolean;
  vote_average: number;
  vote_count: number;
}

interface BelongsToCollection {
  id: number;
  name: string;
  poster_path: string;
  backdrop_path: string;
}

interface Genre {
  id: number;
  name: string;
}

interface ProductionCompany {
  id: number;
  logo_path: string;
  name: string;
  origin_country: string;
}

interface ProductionCountry {
  iso_3166_1: string;
  name: string;
}

interface SpokenLanguage {
  english_name: string;
  iso_639_1: string;
  name: string;
}
```

9. Completar páginas now playing y top rated

- Crear un componente reutilizable `MovieList`:
  - Basarse en la implementación de la página de "Popular" (punto 3).
  - El objetivo es evitar duplicar código: `MovieList` debe recibir las películas como prop y encargarse de renderizarlas.
  - Este componente se usará tanto en la página de "Popular" como en las nuevas páginas ("Now Playing" y "Top Rated").
- Crear dos nuevos servicios:
  - Uno para obtener las películas en cartelera ("Now Playing") usando este endpoint: **https://developer.themoviedb.org/reference/movie-now-playing-list**
  - Otro para obtener las películas mejor calificadas ("Top Rated") usando este endpoint: **https://developer.themoviedb.org/reference/movie-top-rated-list**
- Completar las páginas "Now Playing" y "Top Rated":
  - Basarse en la lógica de la página de "Popular".
  - Cada página debe llamar a su respectivo servicio para traer los datos y mostrarlos usando el componente `MovieList`.
  - La estructura de ambas páginas será similar, cambiando únicamente el servicio que se llama.